Repositórios (Parte 3)

Site: Moodle institucional da UTFPR
Curso: CETEJ34 - Sistemas de Banco de Dados - JAVA_XXX (2024_01)
Livro: Repositórios (Parte 3)

Impresso por: PAULO ROBERTO DOS SANTOS
Data: segunda-feira, 15 jul. 2024, 19:19

Índice

1. Repositórios (Parte 3)

Algo bastante comum no dia a dia dos desenvolvedores que lidam com aplicações e banco de dados é o uso de Stored Procedure. Este recurso é um conjunto de instruções em SQL capaz de ser executado a qualquer momento por uma chamada, a qual pode tanto ser realizada diretamente no SGBD ou via aplicação.

1.1. Procedimentos e Funções

Um Stored Procedure realiza qualquer tipo de ação no banco de dados como um insert, um select ou mesmo a soma entre dois valores. Isto é, depende sempre do que o procedimento foi elaborado para fazer.

Outro ponto importante é que os procedimentos ficam armazenados no banco de dados, por isso é chamado de Stored Procedure (procedimento armazenado).

Na especificação JPA existe uma forma de mapear um Stored Procedure, para sempre que necessário, ele possa ser executado via aplicação. Para isso, é utilizada a anotação **@NamedStoredProcedureQuery**, como o código de exemplo a seguir.

```
@Entity
@NamedStoredProcedureQuery(
   name = "Usuario.soma",
   procedureName = "procedure_soma",
   parameters = {
      @StoredProcedureParameter(
         mode = ParameterMode.IN,
         name = "arg",
         type = Integer.class),
      @StoredProcedureParameter(
         mode = ParameterMode.OUT,
         name = "res",
         type = Integer.class)
   })
public class Usuario {
}
```

Conforme o exemplo apresentado, pode-se notar que existem várias informações incluídas na anotação **@NamedStoredProcedureQuery**. Entre elas, o nome da NamedQuery (**Usuario.soma**), o nome do procedimento no banco de dados (**procedure_soma**) e os dados dos parâmetros de entrada e saída do procedimento.

Mas isso tudo ainda não é o suficiente. Ou seja, para executar o procedimento é necessário que um método da JPA faça a chamada pelo nome da Stored Procedure incluído na anotação. Este método é o **createNamedStoredProcedureQuery()**.

Talvez possa não parecer muito confuso para alguns, ou até confuso demais para outros, mas o importante é que no Spring Data JPA existem diferentes formas para executar um procedimento. Então, vamos ver quais são.

Caso decida manter a anotação @NamedStoredProcedureQuery da JPA, no repositório basta adicionar a assinatura de um método com a anotação @Procedure, conforme o código a seguir:

```
@Procedure ( "procedure_soma" )
Integer procedureSoma(Integer arg);
```

Automaticamente a string na anotação é interpretada como sendo o nome da procedure incluída em **@NamedStoredProcedureQuery**. E assim, o Spring Data vai saber quem precisa ser executado.

Esta informação poderia também ser adicionada ao atributo **procedureName** de **@Procedure**, conforme o exemplo a seguir:

```
@Procedure(procedureName = "procedure_soma")
Integer procedureSoma(Integer arg);
```

Outra forma de referenciar o método a **@NamedStoredProcedureQuery** é pelo nome da NamedQuery, para isso, deve-se usar o atributo **name**, como no exemplo do código a seguir:

```
@Procedure(name =   "Usuario.soma")
Integer procedureSoma(Integer arg);
```

Contudo, se você não quiser usar a anotação da JPA, o processo fica muito mais simples que parece até que está faltando alguma coisa. Para exemplificar, suponha que a Stored Procedure da **Listagem 4.19** foi criada em um banco de dados.

LISTAGEM 4.19: INSTRUÇÃO PARA CRIAR UMA PROCEDURE NO BANCO DE DADOS.

```
CREATE PROCEDURE proc_endereco(IN in_id BIGINT, OUT endereco VARCHAR(254))
READS SQL DATA
BEGIN ATOMIC
   SELECT CONCAT(logradouro, ', ', cidade, ', ', estado, '.')
   INTO endereco
   FROM ENDERECOS WHERE id = in_id;
END;;
```

As informações que precisam ser compreendidas nesta procedure são:

- O nome do procedimento: **proc_endereco**;
- O parâmetro de entrada: **IN in_id BIGINT**;
- O parâmetro de saída: **OUT endereco VARCHAR(254)**;

Essas três informações são importantes para que se possa montar a assinatura do método que vai chamar o procedimento. Sendo assim, é preciso ter um parâmetro do tipo **Long**, um retorno do tipo **String** e é claro, conhecer o nome da procedure.

Agora, vejamos como proceder na aplicação para acessar o procedimento. Na **Listagem 4.20** serão adicionadas duas formas distintas de chamar ou executar o procedimento apresentado.

LISTAGEM 4.20: FORMAS DE FAZER A CHAMADA AO PROCEDIMENTO PELO SPRING DATA JPA.

```
public interface EnderecoRepository extends JpaRepository<Endereco, Long> {
   // código anterior omitido nesta listagem
   @Procedure("proc_endereco")
   String procedureEndereco(Long id);
```

```
   @Procedure
   String proc_endereco(long id);
}
```

Analisando a listagem apresentada, foque primeiro no método **procedureEndereco()**. Observe que no topo de sua assinatura há a anotação @Procedure com o nome dado ao procedimento lá no banco dados. Isto basta para que o Spring Data JPA encontre a Stored Procedure e a execute quando houver uma chamada a este método.

Já em **proc_endereco()**, a **@Procedure** está presente, mas não tem nenhuma instrução referente ao nome do procedimento como parâmetro na anotação. Então, como o Spring Data vai saber qual procedimento no banco de dados ele deveria executar? Ele sabe disso pelo nome dado ao método, que é exatamente o mesmo nome dado a Stored Procedure no banco de dados.

Das duas formas demonstradas, o Spring Data JPA vai encontrar o procedimento diretamente no banco de dados sem que seja necessário usar a anotação da JPA, na classe de entidade, o que torna muito mais simples este processo. Então, fica por sua conta decidir qual é a forma mais adequada para ser usada em seus projetos.

1.2. Parâmetros Nomeados

Por padrão, as consultas JPQL trabalham com parâmetros ordenados ou baseados em posições. Este tipo de instrução leva em consideração a posição do parâmetro na lista de argumentos de um método com a posição do parâmetro na consulta. Veja um exemplo a seguir:

```
@Query( "select c from Contato c "
   + "where idade>= ?1 or nome like ?2")
List<Contato> findByIdadeOuNome(Integer idade, String nome);
```

Analisando a instrução apresentada, observe que ela tem dois argumentos, onde o primeiro é um **Integer** para idade e o segundo é um **String** para o nome.

Esta ordem deve ser referente à ordem dos parâmetros adicionada na JPQL. Quando se faz o uso do **?1** e **?2**, se está dizendo que o **?1** está ligado ao primeiro argumento do método e o **?2** ao segundo argumento. Por isso, esses parâmetros são baseados em posições.

Mas no Spring Data JPA é possível usar a técnica chamada *Named Parameters* ou parâmetros nomeados . Esta técnica substitui os parâmetros baseados em posições por nomes, ou seja, os parâmetros da consulta são vinculados aos argumentos dos métodos por nomes e não por posições.

Para ter uma ideia melhor de como é este processo, veja um exemplo na **Listagem 4.21**.

LISTAGEM 4.21: PARÂMETROS NOMEADOS.

```
public interface ContatoRepository extends JpaRepository<Contato, Long> (
   // código anterior omitido nesta listagem
   @Query("select c from Contato c "
      + "where idade >= : idade or nome like :nome")
   List<Contato> findByIdadeOuNome(@Param("idade") Integer idade,
      @Param("nome") String nome);
}
```

Conferindo o código apresentado, note que na consulta o **?1** e o **?2** foram substituídos por **:idade** e **:nome**. Então, é desta forma que se nomeia os parâmetros dentro da consulta. E para que eles sejam vinculados aos argumentos do método, se utiliza a anotação **@Param** com o nome do parâmetro referente.

É importante destacar que não existe uma regra que defina qual técnica é melhor, se a baseada em posições ou a de parâmetros nomeados. A decisão de qual usar deve ser a de sua preferência.

1.3. Anotação @Modifying

Como já visto até aqui, a anotação **@Query** é um recurso que proporciona diferentes formas de trabalhar com consultas, por exemplo, com JPQL, SQL, e também consultas nomeadas.

O próprio nome da anotação já indica que ela é adequada para consultas. Porém, é permitido alterar esta característica com o uso da anotação **@Modifying**.

A **@Modifying** tem como função indicar que o objetivo original da **@Query** será modificado. Desta forma, é possível adicionar na **@Query** uma JPQL com instrução de **update** ou **delete**.

Confira na **Listagem 4.22** um exemplo dessas operações usando as anotações citadas.

LISTAGEM 4.22: DELETE E UPDATE POR JPQL.

```
public interface EnderecoRepository extends JpaRepository<Endereco, Long> {
   // código anterior omitido nesta listagem
   @Modifying
   @Query("update Endereco e set e.cidade = ?1 where e.id = ?2")
   int updateCidadeById(String cidade, Long id);
```

```
   @Modifying
   @Query("delete from Endereco e where e.id =  ?1")
   int deleteEndereco(Long id);
}
```

Na interface **EnderecoRepository** foram adicionados dois métodos que usam as anotações **@Modifying** e **@Query**. Observe que as instruções JPQL destes métodos são respectivamente um **update** e um **delete** e, essas instruções só são possíveis devido ao uso da anotação **@Modifying**.

Caso a anotação não seja incluída na assinatura de um dos métodos, por exemplo, do **updateCidadeById()**, uma exceção seria lançada ao executá-lo.

Outro aspecto a considerar é o tipo de retorno dos métodos. Cada método apresentado tem como retorno um primitivo **int**. Isto é opcional e pode ser do tipo **void**.

Mas, caso queira ter certeza que a operação foi executada com sucesso, com o uso do **int** o retorno obtido vai ter o valor **do número de registros afetados.**

Importante: Para funcionar o @Modifying, precisamos fazer uso também da anotação **@Transactional** em conjunto para definirmos que a operação de delete ou update ocorra dentro de uma transação, evitando **'TransactionRequiredException'**. Dessa forma a consulta será gerenciada pelo Spring, garantindo a consistência e atomicidade da operação de exclusão. Não se preocupe em saber maiores detalhes sobre controle de transações usando a anotação **@Transactional**, pois esse assunto será abordado no próximo bloco.